AMENDMENTS TO THE SPECIFICATION

Please amend paragraph [0020] as follows:

[0020] FIGS. 13A-13[[C]]F are schematic diagrams showing an example illustrating lazy byteswapping optimization;

Please amend paragraph [0175] as follows:

[0175] FIGS. [[14A]]13A-[[14C]]13F provide an example of lazy byteswapping optimization as described above. The subject code 200 is shown in FIG. 13A of the example as pseudo-code rather than machine code from any particular architecture in order to simplify the example. The subject code 200 describes looping round a number of times, loading a value into register r3, and then storing that value back out. A group block 202 is generated to include two basic blocks, Block 1 and Block 2, illustrated in FIG. 13[[A]]B. Without implementing the lazy byteswapping mechanism, the intermediate representation (IR) generated for the two basic blocks would appear as shown in FIG. 13[[B]]C and FIG 13D. For simplicity, the IR for setting the condition register based on register r1 is not shown in this diagram.

Please amend paragraph [0083] as follows:

[0083] As discussed above, in basic block mode (e.g., Fig. 2), state is passed from one basic block to the next using a memory region which is accessible to all translated code sequences, namely, a global register store 27. The global register store 27 is a repository for abstract registers, each of which corresponds to and emulates the value of a particular subject register or other subject architectural feature. During the execution of translated code 21, abstract registers are held in target registers so that they may participate in instructions. During the execution of ~~translator~~ translated code 21, abstract register values are stored in the global register store 27 or target registers 15.

Please amend paragraph 176 as follows:

[0176] Once the IR for Blocks 1 and 2 have been created, the register definition list is examined looking for byteswaps as the top-level node of the definition. In doing so, it would be discovered that the top-level node 204 for register r3 has been defined as a byteswap (BSWAP). The definition of register r3 is altered to be that of the child of the byteswap node 204, namely the LOAD node 206, where it must be remembered that lazy byteswapping has been invoked. In the IR for Block 2, it can be seen that register r3 is referenced by node 208. Since lazy byteswapping has been invoked in the definition of registerr3, a byteswap must be planted above this reference before it can be used, as shown by the inserted byteswap (BSWAP) node 214 in FIG. 13[[C]]E. In this situation, there are now two consecutive byteswaps, BSWAP node 210 and BSWAP node 214 appearing in the IR for Block 2. Lazy byteswapping optimizations would then fold both of these byteswaps 210 and 214 away such that the byteswap expression would be removed from the IR for both Block 1 and Block 2, as shown in FIG. 13[[C]]E and FIG. 13F. As a result of this lazy byteswapping optimization, the byteswap 204 on the LOAD node 206 (which is in a loop and would be executed multiple times) and the byteswap 210 associated with the store node 212 in Block 2 would be removed from the IR, thus achieving great savings by eliminating these byteswap operations from being generated into target code.